Accumulators in (and Beyond) Generic Groups: Non-Trivial Batch Verification Requires Interaction

Gili Schul-Ganz[⋆] and Gil Segev[⋆]

School of Computer Science and Engineering,
Hebrew University of Jerusalem, Jerusalem 91904, Israel.
{gili.schul,segev}@cs.huji.ac.il

Abstract. We prove a tight lower bound on the number of group operations required for batch verification by any generic-group accumulator that stores a less-than-trivial amount of information. Specifically, we show that $\Omega(t \cdot (\lambda / \log \lambda))$ group operations are required for the batch verification of any subset of $t \geq 1$ elements, where $\lambda \in \mathbb{N}$ is the security parameter, thus ruling out non-trivial batch verification in the standard non-interactive manner.

Our lower bound applies already to the most basic form of accumulators (i.e., static accumulators that support membership proofs), and holds both for known-order (and even multilinear) groups and for unknown-order groups, where it matches the asymptotic performance of the known bilinear and RSA accumulators, respectively. In addition, it complements the techniques underlying the generic-group accumulators of Boneh, Bünz and Fisch (CRYPTO '19) and Thakur (ePrint '19) by justifying their application of the Fiat-Shamir heuristic for transforming their interactive batch-verification protocols into non-interactive procedures.

Moreover, motivated by a fundamental challenge introduced by Aggarwal and Maurer (EUROCRYPT '09), we propose an extension of the generic-group model that enables us to capture a bounded amount of arbitrary non-generic information (e.g., least-significant bits or Jacobi symbols that are hard to compute generically but are easy to compute non-generically). We prove our lower bound within this extended model, which may be of independent interest for strengthening the implications of impossibility results in idealized models.

1 Introduction

Cryptographic accumulators [BdM93], in their most basic form, generate a short commitment to a given set of elements while supporting non-interactive and publicly-verifiable membership proofs. Such accumulators, as well as ones that offer more advanced features (e.g., non-membership proofs, aggregation of proofs

[⋆] Supported by the European Union's Horizon 2020 Framework Program (H2020) via an ERC Grant (Grant No. 714253).

and batch verification) have been studied extensively given their wide applicability to authenticating remotely-stored data (see, for example, [BdM93, ST99, BLL00, CL02, NN98, CJ10, ABC$^+$12, Sla12, MGG$^+$13, CF14, GGM14, PS14] and the references therein).

Known constructions of accumulators can be roughly classified into two categories: hash-based constructions and group-based constructions. Hash-based constructions generate a short commitment via a Merkle tree [Mer87, CHK$^+$08], where the length of the resulting commitment is independent of the number of accumulated elements, and the length of membership proofs and the verification time are both logarithmic in the number of accumulated elements. Group-based constructions, exploiting the *structure* provided by their underlying groups, lead to accumulators in which the length of the commitment, the length of membership proofs and the verification time are all independent of the number of accumulated elements. Such accumulators have been constructed in RSA groups [BP97, CL02, LLX07, Lip12] and in bilinear groups [Ngu05, DT08, CKS09]. In both cases the constructions do not exploit any particular property of the representation of the underlying groups, and are thus generic-group constructions [Nec94, BL96, Sho97, MW98, DK02, Mau05, JS08, JR10, JS13, FKL18].[1]

Accumulators with batch verification. Motivated by recent applications of accumulators to stateless blockchains and interactive oracle proofs [Tod16, BCS16, AHI$^+$17, BSBH$^+$18, BCR$^+$19], Boneh, Bünz and Fisch [BBF19] developed techniques for the aggregation of membership proofs (and even of non-membership proofs) and for their batch verification. Given that hash-based accumulators seem less suitable for offering such advanced features [OWW$^+$20], Boneh et al. exploited the structure provided by RSA groups, and more generally by unknown-order groups such as the class group of an imaginary quadratic number field.

Specifically, Boneh et al. showed that membership proofs and non-membership proofs for any subset of t elements can be aggregated into a single proof whose length is independent of t. Then, by relying on the techniques of Wesolowski [Wes19], they showed that such aggregated proofs can be verified via an *interactive* protocol, where the number of group operations performed by the verifier is nearly independent of t (instead of growing with t in a multiplicative manner as in the verification of t separate proofs). By applying the Fiat-Shamir transform with a hash function that produces random primes, Boneh et al. showed that their interactive verification protocol yields a *non-interactive publicly-verifiable* verification procedure. Analogous results were subsequently obtained in bilinear groups by Thakur [Tha19], who extended the techniques of Boneh et al. and Wesolowski to such groups based on the constructions of Nguyen [Ngu05] and of Damgård and Triandopoulos [DT08].

[1] We note that the RSA-based accumulator hashes the elements into primes before accumulating them. This is captured within the generic-group model since the accumulated elements are provided explicitly as bit-strings (i.e., they are not group elements and therefore such hashing can be performed by generic algorithms). Equivalently, the RSA-based accumulator can be viewed as a generic-group accumulator that accumulates prime numbers.

Non-trivial batch verification vs. interaction. Other than applying the Fiat-Shamir transform for obtaining a non-interactive verification procedure, the constructions of Boneh et al. and Thakur are generic-group constructions, relying on the existing generic-group accumulators in RSA groups [BP97, CL02, LLX07] and in bilinear groups [Ngu05, DT08]. This introduces a substantial gap between generic-group accumulators that support non-trivial batch verification and generic-group accumulators that support only trivial batch verification (i.e., via the verification of individual proofs). Given the key importance of non-interactive verification in most applications that involve accumulators, this leads to the following fundamental question:

Does non-trivial batch verification in generic-group accumulators require interaction?

This question is of significant importance not only from the foundational perspective of obtaining a better understanding of the feasibility and efficiency of supporting advanced cryptographic features, but also from the practical perspective. Specifically, following Wesolowski [Wes19], Boneh et al. implement the Fiat-Shamir transform using a hash function that produces random primes. As discussed by Boneh et al. [BBF19] and by Ozdemir, Wahby, Whitehat and Boneh [OWW^{+}20], who proposed various potential realizations for such a hash function, this affects the efficiency, the correctness and the security of the resulting accumulator. More generally, and even when implementing the Fiat-Shamir transform via any standard hash function, in many cases the transformation violates the elegant algebraic structure of the underlying interactive protocol, leading to potentially-substantial overheads when implemented within larger systems (e.g., systems that rely on efficient algebraic proof systems).

1.1 Our Contributions

We prove a tight lower bound on the number of group operations performed during batch verification by any generic-group accumulator that uses less-than-trivial space. In particular, we show that no such accumulator can support non-trivial batch verification in the standard non-interactive manner. Our lower bound applies already to the most basic form of accumulators (i.e., static accumulators that support membership proofs), and holds both for known-order (and even multilinear) groups and for unknown-order groups, where it matches the asymptotic performance of the known bilinear and RSA accumulators, respectively.[2]

Moreover, motivated by a fundamental challenge introduced by Aggarwal and Maurer [AM09]), we propose an extension of the generic-group model that enables us to capture a bounded amount of arbitrary non-generic information (e.g., least significant bits or Jacobi symbols that are hard to compute generically but are easy to compute non-generically [AM09, JS13]). We prove our lower bound within

[2] Our results hold also for the more restrictive notion of vector commitments [CF13, BBF19, LM19], which provide the same functionality as accumulators, but for ordered lists instead of sets.

this extended model, where we measure efficiency in terms of the number of group operations and in terms of the amount of non-generic information. This extension of the generic-group model may be of independent interest for strengthening the implications of impossibility results in idealized models.

In what follows we state our results somewhat informally in order to avoid introducing the entire list of parameters with which we capture the efficiency of generic-group accumulators (we refer the reader to Section 2.2 for our formal definitions). Here we will focus on the following three main parameters, n_{Acc}, ℓ_{Acc} and q, that are associated with an accumulator $\mathcal{ACC}$, where we denote by $\lambda \in \mathbb{N}$ the security parameter:

- $n_{\mathsf{Acc}}(\lambda, k)$ and $\ell_{\mathsf{Acc}}(\lambda, k)$ are the number of group elements and the bit-length of the explicit string, respectively, stored by the accumulator when accumulating k elements.
- $q(\lambda, t, k)$ is the number of group-operation queries issued by the accumulator's verification procedure when verifying a membership proof for t out of k elements.

Our main result. Our main result is a tight bound on the trade-off between the amount of information that an accumulator stores when accumulating $k \geq 1$ elements and its number of group operations when verifying a membership proof for $1 \leq t \leq k$ elements. We prove that this number of group operations must scale multiplicatively with t, thus ruling out non-trivial batch verification. This is captured by the following theorem which applies both to known-order groups and to unknown-order groups.[3]

Theorem 1.1 (Simplified). *For any generic-group accumulator $\mathcal{ACC}$ over a domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ it holds that*

$$q(\lambda, t, k) = \Omega\left(t \cdot \frac{\log_2 \binom{|\mathcal{X}_\lambda|}{k} - \left[n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}} + 1) + \ell_{\mathsf{Acc}}\right]}{k} \cdot \frac{1}{\log \lambda}\right)$$

for all sufficiently large $\lambda \in \mathbb{N}$. In particular, if $|\mathcal{X}_\lambda| = 2^{\Omega(\lambda)}$ then for any $0 < \epsilon < 1$ either

$$n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}} + 1) + \ell_{\mathsf{Acc}} \geq (1 - \epsilon) \cdot \log_2 \binom{|\mathcal{X}_\lambda|}{k}$$

or

$$q(\lambda, k, t) = \Omega\left(t \cdot \frac{\epsilon\lambda}{\log \lambda}\right).$$

For interpreting our main theorem, note that $\log_2 \binom{|\mathcal{X}_\lambda|}{k}$ is the expected number of bits required for an exact representation of k elements, and that $n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}} +$

[3] We note that all logarithms in this paper are to the base of 2, which we omit whenever used within asymptotic expressions in a multiplicative manner.

$1) + \ell_{\mathsf{Acc}}$ is the amount of information that is actually stored by a generic-group accumulator from its verification algorithm's point of view: $n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}}+1)$ bits of information resulting from the equality pattern among the n_{Acc} stored group elements, and ℓ_{Acc} additional explicit bits of information. Thus, the expression

$$\frac{\log_2 \binom{|\mathcal{X}_\lambda|}{k} - \left[n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}} + 1) + \ell_{\mathsf{Acc}}\right]}{k}$$

captures the average information loss per accumulated element. Our theorem shows that as long as the amount of information stored by an accumulator is bounded away from the information-theoretic amount that is required for an exact representation, then non-trivial batch verification is impossible.

Our lower bound on the efficiency of batch verification matches the performance of the RSA accumulator considered by Boneh et al. [BBF19] in which $n_{\mathsf{Acc}} = 1$ and $\ell_{\mathsf{Acc}} = 0$ (i.e., the accumulator stores just a single group element), and $|\mathcal{X}_\lambda| = 2^{\Omega(\lambda)}$. In this accumulator, batch verification of t elements can be computed via a single exponentiation in the group $\mathbb{Z}_N^*$, where the exponent is the product of t numbers, each of which is of length λ bits. Since the order of the group is unknown, it seems that the exponent cannot be reduced modulo the group order prior to the exponentiation, and therefore this computation requires $\Omega(t \cdot \lambda)$ group operations, or $\Omega\left(t \cdot \frac{\lambda}{\log \lambda}\right)$ group operations with preprocessing [BGM⁺92] – thus matching our lower bound.[4]

Moreover, we show that our result holds even in the generic d-linear group model for any $d \geq 2$. We generalize Theorem 1.1 and similarly show that $\Omega\left(t \cdot \frac{\lambda}{\log \lambda} \cdot \frac{1}{d}\right)$ group operations are required for batch verifying a membership proof for t elements. This matches the performance of the bilinear accumulator constructed by Nguyen [Ngu05] in which $n_{\mathsf{Acc}} = 1$ and $\ell_{\mathsf{Acc}} = 0$ (i.e., the accumulator stores just a single group element), and $|\mathcal{X}_\lambda| = 2^{\Omega(\lambda)}$. In this accumulator, trivial batch verification of t elements consists of computing t exponentiations, translating to $\Omega(t \cdot \lambda)$ group operations, or to $\Omega\left(t \cdot \frac{\lambda}{\log \lambda}\right)$ group operations with preprocessing as above. This once again matches our lower bound, showing that trivial batch verification is indeed optimal for this accumulator.

Beyond generic groups. Lower bounds in idealized models shed substantial insight on our understanding of a wide range of both hardness assumptions and cryptographic constructions. For example, such lower bounds apply to a wide range of algorithms and constructions, and thus help directing cryptanalytic efforts and candidate constructions away from generic impossibility results.

However, despite their importance, a major drawback of such lower bounds is clearly their restriction to idealized models. This drawback was discussed by Aggarwal and Maurer [AM09], noting that there are certain computations that are hard with respect to generic algorithms but are extremely simple with

[4] The additional information resulting from such preprocessing can be included with the information stored by the accumulator. This amount of information is independent of the number of accumulated elements, and thus does not influence our result.

respect to non-generic ones. Two important examples for such computations are computing the least significant bit [AM09] or the Jacobi symbol of a random group element [JS13]. Motivated by this major drawback, Aggarwal and Maurer proposed the problem of considering more general and realistic models where all algorithms are given access, for example, to least significant bits or Jacobi symbols of elements.

Addressing the challenge introduced by Aggarwal and Maurer, we show that our techniques are applicable even in an extended model that enables us to capture a bounded amount of non-generic information. Specifically, for any family $\mathit{\Phi}$ of predicates $\phi(\cdot,\cdot)$ that take as input the group order and a group element, we equip all algorithms with access to an oracle that responds to $\mathit{\Phi}$-queries: On input a query of the form $(\phi, \widehat{x})$, where $\phi \in \mathit{\Phi}$ and $\widehat{x}$ is an implicit representation of a group element x, the oracle returns $\phi(N, x)$ where N is the order of the group. We refer to this extension as the $\mathit{\Phi}$-augmented generic-group model, and note that the family $\mathit{\Phi}$ may be tailored to the specific non-generic structure of any underlying group. This model, allowing a bounded amount of non-generic information, can be viewed as an intermediate model between the generic-group model that does not allow any non-generic information, and the algebraic-group model [FKL18, MTT19, AHK20, FPS20] that allows direct access to the representation of the underlying group.

At a high-level, we prove that our result still holds for any family $\mathit{\Phi}$ of polynomially-many predicates (in particular, it still holds for the case $|\mathit{\Phi}| = 2$ that enables to compute least significant bits and Jacobi symbols). More specifically, letting $q(\lambda, t, k)$ denote the number of group-operation queries and $\mathit{\Phi}$-queries issued by an accumulator's verification procedure when verifying a membership proof for t out of k elements, and considering also predicate families $\mathit{\Phi}$ of super-polynomial size, we prove the following theorem (which again applies both to known-order groups and to unknown-order groups).

Theorem 1.2 (Simplified). *For any predicate family $\mathit{\Phi}$ and for any $\mathit{\Phi}$-augmented generic-group accumulator $\mathcal{ACC}$ over a domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ it holds that*

$$q(\lambda, t, k) = \Omega\left(t \cdot \frac{\log_2 \binom{|\mathcal{X}_\lambda|}{k} - \left[n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}} + 1) + \ell_{\mathsf{Acc}}\right]}{k} \cdot \frac{1}{\log \lambda + \log |\mathit{\Phi}|}\right)$$

for all sufficiently large $\lambda \in \mathbb{N}$. In particular, if $|\mathcal{X}_\lambda| = 2^{\Omega(\lambda)}$ then for any $0 < \epsilon < 1$ either

$$n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}} + 1) + \ell_{\mathsf{Acc}} \geq (1 - \epsilon) \cdot \log_2 \binom{|\mathcal{X}_\lambda|}{k}$$

or

$$q(\lambda, k, t) = \Omega\left(t \cdot \frac{\epsilon\lambda}{\log \lambda + \log |\mathit{\Phi}|}\right).$$

It should be noted that our result in this augmented model do not contradict the highly-efficient non-interactive batch verification procedures of the accumulators constructed by Boneh, Bünz and Fisch [BBF19] and by Thakur [Tha19].

Their verification procedures are obtained by applying the (non-generic) Fiat-Shamir transform to interactive verification protocols. Although our augmented model does allow any predicate family Φ, the trade-off resulting from Theorem 1.2 becomes meaningless when instantiated with the parameters that are required for accommodating the Fiat-Shamir transform.

For example, it is possible to consider a predicate family Φ that consists of predicates ϕ_i that output the i-th output bit of any given collection of hash functions. However, within our model, the family Φ has to be fixed ahead of time, whereas the soundness of the Fiat-Shamir transform relies on the hash function being completely random. This means that realizing the Fiat-Shamir transform within our augmented model requires including such a predicate $\phi_{h,i}$ for every function h mapping group elements to, say, λ bits, as this would then enable sampling a random function. However, in this case, the size $|\Phi|$ of the family Φ becomes too large for our trade-off to be meaningful. An additional example is a predicate family Φ that consists of predicates ϕ_i that directly output the i-th bit of a group element. Applying these predicates to all group elements in the view of the verification algorithm increases the number q of queries that are issued by the verification algorithm at least by a multiplicative factor of λ (i.e., λ queries for each group element), and then once again our trade-off is no longer meaningful – and thus does not contradict the known non-generic constructions.

1.2 Overview of Our Approach

The framework. We prove our result within the generic-group model introduced by Maurer [Mau05], which together with the incomparable model introduced by Shoup [Sho97], seem to be the most commonly used approaches for capturing generic-group computations. At a high level, in both models algorithms have access to an oracle for performing the group operation and for testing whether two group elements are equal. The difference between the two models is in the way that algorithms specify their queries to the oracle. In Maurer's model algorithms specify their queries by pointing to two group elements that have appeared in the computation so far (e.g., the 4th and the 7th group elements), whereas in Shoup's model group elements have an explicit representation (sampled uniformly at random from the set of all injective mappings from the group to sufficiently long strings) and algorithms specify their queries by providing two strings that have appeared in the computation so far as encodings of group elements.

Jager and Schwenk [JS08] proved that the complexity of any computational problem that is defined in a manner that is independent of the representation of the underlying group (e.g., computing discrete logarithms) in one model is essentially equivalent to its complexity in the other model. However, not all generic cryptographic constructions are independent of the underlying representation.

More generally, these two models are rather incomparable. On one hand, the class of cryptographic schemes that are captured by Maurer's model is a subclass of that of Shoup's model – although as demonstrated by Maurer his model still captures all schemes that only use the abstract group operation and test whether two group elements are equal. On the other hand, the same holds also for the class of adversaries, and thus in Maurer's model we have to break the security of a given scheme using an adversary that is more restricted when compared to adversaries in Shoup's model. We refer the reader to Section 2.1 for a formal description of Maurer's generic-group model.[5]

Generic-group accumulators. A generic-group accumulator $\mathcal{ACC}$ consists of three algorithms, denoted Setup, Prove and Vrfy. Informally (and very briefly), the algorithm Setup receives as input a set $X \subseteq \mathcal{X}$ of elements to accumulate and produces a representation Acc together with a secret state, where $\mathcal{X}$ is the universe of all possible elements. The algorithm Prove receives as input the secret state and a set $S \subseteq X$, and outputs a membership proof π, which can then be verified by the algorithm Vrfy. Note that the case $|S| = 1$ captures standard verification of individual elements, whereas the case $|S| > 1$ captures batch verification (i.e., simultaneous verification of sets of elements). Each of these algorithms may receive as input and return as output a combination of group elements and explicit strings.

We consider the standard notion of security for accumulators when naturally extended to consider batch verification. That is, we consider an adversary who specifies a set $X \subseteq \mathcal{X}$ of elements, receives an accumulator Acc that is honestly generated for X, and can then ask for honestly-generated membership proofs for sets $S \subseteq X$ (in fact, the adversary we present for proving our result does not require such *adaptive* and *post-challenge* access to honestly-generated proofs). Then, the adversary aims at outputting a pair (S^*, π^*) that is accepted by the verification algorithm as a valid membership proof for the set S^* with respect to the accumulator Acc although $S^* \nsubseteq X$. We refer the reader to Section 2.2 for formal definitions.

From capturing information loss to exploiting it. We prove our result by presenting a generic-group adversary that attacks any generic-group accumulator. Our attacker is successful against any accumulator that does not satisfy the trade-off stated in Theorem 1.1 between the amount of information that the accumulator stores and the number of group-operation queries issued by its verification algorithm. The main idea underlying our approach can be summarized via the following two key steps:

- **Step I: Capturing the information loss.** We identify and account for the total amount of information on a random set X of accumulated elements from the point of view of a generic-group verification algorithm.
- **Step II: Exploiting the information loss.** We show that any gap between this amount of information and the amount of information that is required for an exact representation of such a set X can be exploited by a generic-group attacker for generating a false batch-membership proof.

[5] In fact, we consider two different flavors of Maurer's model, for capturing both known-order and unknown-order groups. The reader is referred to Section 2.1 for an in-depth discussion of these two flavors and of the extent to which each of them captures group-based cryptographic constructions.

In what follows we elaborate on these two steps, first focusing on our main result and then discussing its extensions. Let $\mathcal{ACC} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy})$ be a generic-group accumulator, and consider the view of its verification algorithm Vrfy on input an accumulator Acc, a set S, and a membership proof π for the fact that all elements of S have been accumulated within Acc. For simplicity, we assume here that Acc and π consist only of group elements, and we note that our proof in fact considers the more general case where they may consist of both group elements and explicit strings. Then, the view of the verification algorithm consists of the following ingredients:

– The accumulator Acc consists of group elements, and therefore the verification algorithm essentially only observes the equality pattern among these elements, and does not observe the elements themselves. This enables us to upper bound the amount of information provided by Acc by upper bounding the number of possible equality patterns among the group elements that are included in Acc.
– Once the computation starts, the verification algorithm generates a sequence of group-operation queries, where each such query is specified by pointing to two group elements that have appeared in the computation so far (we allow the verification algorithm to issue all possible equality queries). The following two observations enable us to upper bound the amount of information provided by this computation by upper bounding the number of possible query patterns that the verification algorithm observes, together with the number of possible equality patterns among the group elements included in the proof π and among the responses to the queries: (1) There are only polynomially-many possibilities for the two pointers included in each query (since queries are specified by pointing to two group elements that have appeared in the computation so far), and (2) we can effectively upper bound the number of possible query patterns induced by the proof and the responses using the number of queries issued by the verification algorithm instead of using the length of the proof π (which may be significantly larger).

This accounts for the total amount of information that is available to the verification algorithm from a single execution. However, different executions of the verification algorithm may be highly correlated via Acc and via the membership proofs (which are all generated from the secret state). Therefore, in order to capture the total amount of information that is available on the entire accumulated set X, our attacker $\mathcal{A}$ gathers this information as follows. First, it chooses a random set $X \subseteq \mathcal{X}$ of k elements for which the setup algorithm Setup will honestly generate an accumulator Acc. Then, $\mathcal{A}$ partitions X into subsets of size t, and asks for an honestly-generated batch-membership proof for each such subset. Next, $\mathcal{A}$ executes the verification algorithm to verify each of these k/t proofs, and records the above information for all of the subsets.

At this point we show that the information recorded from these k/t batch verifications must be at least the amount of information that is required for representing a random set X of size k. This is done by proving that, with high probability over the choice of X, this information can be exploited for forging a batch-membership proof for a set $S^* \nsubseteq X$ of size t. The most subtle part of our proof is in tailoring the set S^* and its false proof in a manner that is indistinguishable to the verification algorithm from those of at least one of the k/t subsets of X, and we refer the reader to Section 3 for the details of this part of our attack.

Extensions. As discussed above, we extend our result to accumulators in generic d-linear groups and to accumulators that rely on a bounded amount of non-generic information. Both of these extensions essentially rely on the same basic idea, which is the observation that each query issued by the verification algorithm can be fully represented in a somewhat succinct manner. Specifically, each such query is determined by: (1) Pointers to its inputs (where the number of inputs may now range from 2 to d), (2) the type of query (e.g., addition or subtraction queries in $\mathbb{Z}_N$, multilinear queries, or any other type of non-generic query $\phi \in \Phi$), and (3) the contribution of its response to the equality pattern among all group elements involved in the computation, or the contribution of its explicit response to the overall amount of information in the case of non-generic queries. For each of these two extensions, we first adapt our proof to identify and account for the total amount of information on a random set X of accumulated elements from the verification algorithm's point of view. Then, we accordingly adapt our tailored set S^* and its false proof in a manner that remains indistinguishable to the verification algorithm even when equipped with more expressive queries.

1.3 Related Work

In addition to the above-discussed motivation underlying our work, the problem we consider can be viewed as inspired by a long line of research on proving efficiency trade-offs for various primitives that are constructed in a black-box manner in the standard model (see, for example, [KST99, GGK$^+$05, BM07, Wee07, BM09, HK10, HHR$^+$15] and the many references therein). Despite the similarity in terms of the goal of proving efficiency trade-offs, there are several fundamental differences between this line of research and our work, as we now discuss.

Conceptually, results in this line of research provide lower bounds for constructions that are based on specific and somewhat weak assumptions, such as the existence of one-way functions or permutations. Our work provides a lower bound for any generic-group scheme, capturing assumptions that seem significantly stronger than the existence of minimal cryptographic primitives. As a consequence, our lower bound applies to a wide variety of practical constructions, instead of somewhat theoretical constructions that are based on minimal assumptions.

Generally speaking, it is more challenging to prove lower bounds for schemes in the generic-group model when compared to lower bounds for black-box constructions based on minimal assumptions. One-way functions or permutations are typically modeled via random functions or permutations, which admit very little structure that can be utilized by cryptographic constructions. This stands in complete contrast to generic-group constructions that exploit the algebraic structure of the underlying groups. The prime example for this substantial gap is the fact that key-agreement protocols do not exist relative to a random function or permutation, but do exist based on the decisional Diffie-Hellman assumption and thus in the generic-group model [DH76, IR89, BM09].

Technically, out of this long line of research, the result that is closest to the problem we consider is that of Horvitz and Katz [HK10]. They proved a lower bound on the efficiency of statistically-binding commitment schemes based on one-way permutations, in terms of the number of invocations of the one-way permutation during the commit stage. In addition to the above-discussed differences between this line of research and our work, here we would like to point out two more differences. First, Horvitz and Katz proved a lower bound for a primitive with statistical soundness, whereas we consider a primitive with standard computational soundness[6]. Second, and much more crucial, they proved a lower bound on the efficiency of the *commit* stage, whereas we prove a lower bound on the efficiency of *verification*. This is especially crucial given that accumulators can be viewed as commitments with short local openings, and thus in general a lower bound on the efficiency of the commit stage does not seem to imply any meaningful lower bound on the efficiency of the decommit stage.

1.4 Paper Organization

The remainder of this paper is organized as follows. First, in Section 2 we present the basic notation used throughout the paper, and formally describe the framework of generic-group accumulators. In Section 3 we prove our main result in the generic-group model, and in Section 4 we briefly discuss several open problems that arise from this work. Due to space limitations we refer the reader to the full version of this work for the extension of our result to the generic multilinear-group model and to its extension beyond the generic-group model.

2 Preliminaries

In this section we present the basic notions and standard cryptographic tools that are used in this work. For a distribution X we denote by $x \leftarrow X$ the process of sampling a value x from the distribution X. Similarly, for a set $\mathcal{X}$ we denote by $x \leftarrow \mathcal{X}$ the process of sampling a value x from the uniform distribution over $\mathcal{X}$. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \dots, n\}$. For a vector $v \in \mathcal{X}^k$, where $\mathcal{X}$ is a set and $k \in \mathbb{N}$, and for any $j \in [k]$, we denote by $(v)_j$ the jth coordinate of v. For a set $\mathcal{J} \subseteq \mathbb{Z}$ and an integer $i \in \mathbb{Z}$ we let $i + \mathcal{J} = \{i + j | j \in \mathcal{J}\}$. A function $\nu : \mathbb{N} \to \mathbb{R}^+$ is *negligible* if for any polynomial $p(\cdot)$ there exists an integer N such that for all $n > N$ it holds that $\nu(n) \leq 1/p(n)$.

[6] When interpreted in our setting of the generic-group model (where algorithms are unbounded in their internal computation), computational soundness considers adversaries that issue a polynomial bounded of group-operation queries, whereas statistical soundness considers adversaries that may issue an unbounded number of such queries.

2.1 Generic Groups and Algorithms

We prove our results within the generic-group model introduced by Maurer [Mau05]. We consider computations in cyclic groups of order N (all of which are isomorphic to $\mathbb{Z}_N$ with respect to addition modulo N), for a λ-bit integer N that is generated by an order-generation algorithm $\mathsf{OrderGen}(1^\lambda)$, where $\lambda \in \mathbb{N}$ is the security parameter (and N may or may not be prime).

When considering such groups, each computation in Maurer's model is associated with a table $\mathbf{B}$. Each entry of this table stores an element of $\mathbb{Z}_N$, and we denote by V_i the group element that is stored in the ith entry. Generic algorithms access this table via an oracle $\mathcal{O}$, providing black-box access to $\mathbf{B}$ as follows. A generic algorithm $\mathcal{A}$ that takes d group elements as input (along with an optional bit-string) does not receive an explicit representation of these group elements, but instead, has oracle access to the table $\mathbf{B}$, whose first d entries store the $\mathbb{Z}_N$ elements corresponding to the d group element in $\mathcal{A}$'s input. That is, if the input of an algorithm $\mathcal{A}$ is a tuple $(g_1, \ldots, g_d, x)$, where $g_1, \ldots, g_d$ are group elements and x is an arbitrary string, then from $\mathcal{A}$'s point of view the input is the tuple $(\widehat{g_1}, \ldots, \widehat{g_d}, x)$, where $\widehat{g_1}, \ldots, \widehat{g_d}$ are pointers to the group elements $g_1, \ldots, g_d$ (these group elements are stored in the table $\mathbf{B}$), and x is given explicitly. All generic algorithms in this paper will receive as their first input a generator of the group; we capture this fact by always assuming that the first entry of $\mathbf{B}$ is occupied by $1 \in \mathbb{Z}_N$, and we will sometimes forgo noting this explicitly. The oracle $\mathcal{O}$ allows for two types of queries:

- **Group-operation queries:** On input $(i, j, \circ)$ for $i, j \in \mathbb{N}$ and $\circ \in \{+, -\}$, the oracle checks that the ith and jth entries of the table $\mathbf{B}$ are not empty, computes $V_i \circ V_j \bmod N$ and stores the result in the next available entry. If either the ith or the jth entries are empty, the oracle ignores the query.
- **Equality queries:** On input $(i, j, =)$ for $i, j \in \mathbb{N}$, the oracle checks that the ith and jth entries of the table $\mathbf{B}$ are not empty, and then returns 1 if $V_i = V_j$ and 0 otherwise. If either the ith or the jth entries are empty, the oracle ignores the query.

In this paper we consider interactive computations in which multiple algorithms pass group elements (as well as non-group elements) as inputs to one another. This is naturally supported by the model as follows: When a generic algorithm $\mathcal{A}$ outputs k group elements (along with a potential bit-string σ), it outputs the indices of k (non-empty) entries in the table $\mathbf{B}$ (together with σ). When these outputs (or some of them) are passed on as inputs to a generic algorithm $\mathcal{C}$, the table $\mathbf{B}$ is re-initialized, and these values (and possibly additional group elements that $\mathcal{C}$ receives as input) are placed in the first entries of the table. Additionally, we rely on the following conventions:

1. Throughout the paper we refer to values as either "explicit" ones or "implicit" ones. Explicit values are all values whose representation (e.g., binary strings of a certain length) is explicitly provided to the generic algorithms under consideration. Implicit values are all values that correspond to group elements

and that are stored in the table $\mathbf{B}$ – thus generic algorithms can access them only via oracle queries. We will sometimes interchange between providing group elements as input to generic algorithms implicitly, and providing them explicitly. Note that moving from the former to the latter is well defined, since a generic algorithm $\mathcal{A}$ that receives some of its input group elements explicitly can always simulate the computation as if they were received as part of the table $\mathbf{B}$.

2. For a group element g, we will differentiate between the case where g is provided explicitly and the case where it is provided implicitly via the table $\mathbf{B}$, using the notation g in the former case, and the notation $\widehat{g}$ in the latter. Additionally, we extend this notation to a vector v of group elements, which may be provided either explicitly (denoted v) or implicitly via the table $\mathbf{B}$ (denoted $\widehat{v}$).

Known-order vs. unknown-order generic groups. We consider two flavors of generic groups: groups of known orders and groups of unknown orders. In the case of known-order groups, as discussed above we prove our results within Maurer's generic-group model [Mau05] that lets all algorithms receive the order of the underlying group as an explicit input.

In the case of unknown-order groups, we prove our results in a natural variant of Maurer's model by following the approach of Damgård and Koprowski [DK02]. They considered a variant of Shoup's "random-encoding" model [Sho97] where the order of the underlying group is not included as an explicit input to all algorithms (still, however, the corresponding order-generation algorithm $\mathsf{OrderGen}$ is publicly known). We consider the exact same variant of Maurer's model (i.e., Maurer's model where the order of the underlying group is not included as an explicit input to all algorithms) for proving our results for unknown-order groups.

The known-order and unknown-order flavors of generic groups are incomparable for analyzing the security of cryptographic constructions. In the known-order variant, constructions can explicitly rely on the order of the underlying group, but this holds for attackers as well. In the unknown-order variant, neither constructions or attackers can explicitly rely on the order of the underlying group.

Finally, it should be noted that these two flavors of generic groups seem to somewhat differ in the extents in which they capture group-based constructions of cryptographic primitives. While the known-order flavor seems to capture quite accurately generic computations in prime-order cyclic groups and multilinear groups, the unknown-order flavor seems somewhat less accurate in capturing generic computations in RSA groups. Specifically, in the unknown-order flavor, the order of the underlying group is hidden in an information-theoretic manner and generic algorithm are unbounded in their internal computation. However, in "natural" RSA-based constructions, the order of the underlying group is only computationally hidden (i.e., the modulus $N = P \cdot Q$ is known but the order of the multiplicative group $\mathbb{Z}_N^*$ is unknown based on the factoring assumption), and algorithms are polynomially-bounded in their computation.

Addressing these differences, Aggarwal and Maurer [AM09] proposed the incomparable generic-ring model, where algorithms are provided with the modulus

N but are restricted in their computation. A interesting open problem for future research is whether or not our techniques extend to other idealized models such as the generic-ring model. Despite any potential differences between the various models, impossibility results in any idealized model direct cryptanalytic efforts and candidate constructions away from generic impossibility results, and serve as a necessary step towards proving such results within less-idealized models.

2.2 Generic-Group Accumulators

For concreteness, we frame the following definition for known-order generic groups, noting that the analogous definition for unknown-order generic groups is obtained by not providing the order N of the underlying group as an input to any of the algorithms. Our definition is parameterized by 5 functions corresponding to the measures of efficiency that are considered in our work, and we refer the reader to Table 1 for the list of all the parameters used in the following definition.

Definition 2.1. *A generic-group* $(n_{\mathsf{Acc}}, \ell_{\mathsf{Acc}}, n_\pi, \ell_\pi, q)$*-accumulator over a domain* $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ *is a triplet* $\mathcal{ACC} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy})$ *of generic algorithms defined as follows:*

- *The algorithm* Setup *is a probabilistic algorithm that receives as input the security parameter* $\lambda \in \mathbb{N}$*, the group order* N *and a set* $X \subseteq \mathcal{X}_\lambda$*. It outputs an accumulator* $\mathsf{Acc} = (\widehat{\mathsf{Acc}_\mathsf{G}}, \mathsf{Acc}_\mathsf{str})$ *and a state* $\mathsf{state} \in \{0,1\}^*$*, where* Acc_G *is a sequence of* $n_{\mathsf{Acc}}(\lambda, |X|)$ *group elements, and* $\mathsf{Acc}_\mathsf{str} \in \{0,1\}^{\ell_{\mathsf{Acc}}(\lambda, |X|)}$*.*
- *The algorithm* Prove *is a probabilistic algorithm that receives as input an accumulator* Acc*, a state* $\mathsf{state} \in \{0,1\}^*$ *and a set* $S \subseteq \mathcal{X}_\lambda$*, and outputs a proof* $\pi = (\widehat{\pi_G}, \pi_\mathsf{str})$*, where* π_G *is a sequence of* $n_\pi(\lambda, |S|, k)$ *group elements,* $\pi_\mathsf{str} \in \{0,1\}^{\ell_\pi(\lambda, |S|, k)}$ *is an explicit string, and* k *is the number of elements that have been accumulated by* Acc*.*
- *The algorithm* Vrfy *is a deterministic algorithm that receives as input an accumulator* Acc*, a set* $S \subseteq \mathcal{X}_\lambda$ *and a proof* π*, issues an arbitrary number of* equality *queries and at most* $q(\lambda, |S|, k)$ group-operation *queries and outputs a bit* $b \in \{0,1\}$*, where* k *is the number of elements that have been accumulated by* Acc*. Note that we do not restrict the number of equality queries that are issued by the verification algorithm and this only makes our lower bound stronger (i.e., our lower bound on the number of group-operation queries holds even for accumulators in which the verification algorithm issues all possible equality queries).*

Correctness. The correctness requirement for this most basic form of accumulators is quite natural: For any set $X \subseteq \mathcal{X}_\lambda$ of accumulated elements, any membership proof that is generated by the algorithm Prove for any set $S \subseteq X$ should be accepted by the algorithm Vrfy. More formally:

Definition 2.2. *A generic-group accumulator* $\mathcal{ACC} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy})$ *over a domain* $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ *is correct with respect to an order-generation algorithm*

λ	The security parameter
$k(\lambda)$	The number of accumulated elements (i.e., $k = \|X\|$)
$t(\lambda)$	The number of elements for which a batch membership proof is generated and then verified (i.e., $t = \|S\|$ where $S \subseteq X$)
$n_{\mathsf{Acc}}(\lambda, k)$	The number of group elements produced by Setup when accumulating a set of k elements
$\ell_{\mathsf{Acc}}(\lambda, k)$	The bit-length of the explicit string produced by Setup when accumulating a set of k elements
$n_{\pi}(\lambda, t, k)$	The number of group elements produced by Prove when proving membership of a set of t elements out of k accumulated elements
$\ell_{\pi}(\lambda, t, k)$	The bit-length of the explicit string produced by Prove when proving membership of a set of t elements out of k accumulated elements
$q(\lambda, t, k)$	The number of *group-operation* queries issued by Vrfy when verifying a membership proof for a set of t elements out of k accumulated elements (we prove our lower bound even for verification algorithms that issue an arbitrary number of *equality* queries)

Table 1. The parameters considered in Definition 2.1.

OrderGen *if for any $\lambda \in \mathbb{N}$ and for any two sets $S \subseteq X \subseteq \mathcal{X}_\lambda$, it holds that*

$$\Pr\left[\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S, \pi) = 1\right] = 1$$

where $N \leftarrow \mathsf{OrderGen}(1^\lambda)$, $(\mathsf{Acc}, \mathsf{state}) \leftarrow \mathsf{Setup}^{\mathcal{O}}(\lambda, N, X)$ and $\pi \leftarrow \mathsf{Prove}^{\mathcal{O}}(\mathsf{Acc}, \mathsf{state}, S)$, and the probability is taken over the internal randomness of all algorithms.

Security. We extend the standard notion of security for accumulators to consider batch verification (i.e., supporting the simultaneous verification of sets of elements instead of individual elements). Our notion of security considers an adversary who specifies a set $X \subseteq \mathcal{X}_\lambda$ of elements, receives an accumulator Acc that is honestly generated for X, and can then ask for honestly-generated membership proofs for sets $S \subseteq X$. Then, the adversary aims at outputting a pair (S^*, π^*), where $S^* \subseteq \mathcal{X}_\lambda$, that is accepted by the verification algorithm as a valid membership proof for the set S^* with respect to the accumulator Acc although $S^* \nsubseteq X$.

Definition 2.3. *A generic-group accumulator $\mathcal{ACC} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy})$ over a domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ is secure with respect to an order-generation algorithm* OrderGen *if for any algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ that issues a polynomial number of queries there exists a negligible function $\nu(\lambda)$ such that*

$$\Pr\left[\mathsf{Expt}_{\mathcal{ACC},\mathcal{A}}(\lambda) = 1\right] \leq \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$, where the experiment $\mathsf{Expt}_{\mathcal{ACC},\mathcal{A}}(\lambda)$ is defined as follows:

1. $N \leftarrow \mathsf{OrderGen}(1^\lambda)$.
2. $X \leftarrow \mathcal{A}_0^{\mathcal{O}}(1^\lambda, N)$ *where* $X \subseteq \mathcal{X}_\lambda$.
3. $(\mathsf{Acc}, \mathsf{state}) \leftarrow \mathsf{Setup}^{\mathcal{O}}(1^\lambda, N, X)$.
4. $(S^*, \pi^*) \leftarrow \mathcal{A}_1^{\mathcal{O}, \mathsf{Prove}^{\mathcal{O}}(\mathsf{Acc}, \mathsf{state}, \cdot)}(1^\lambda, N, \mathsf{Acc})$ *where* $S^* \subseteq \mathcal{X}_\lambda$.
5. *If* $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S^*, \pi^*) = 1$ *and* $S^* \not\subseteq X$ *then output* 1*, and otherwise output* 0*.*

Note that the above definition provides the algorithm $\mathcal{A}_1$ with *adaptive* and *post-challenge* access to the oracle $\mathsf{Prove}^{\mathcal{O}}(\mathsf{Acc}, \mathsf{state}, \cdot)$. In fact, the adversaries we present for proving our results do not require such a strong form of access to honestly-generated proofs. Specifically, already our algorithm $\mathcal{A}_0$ can specify a list of queries to this oracle, in a completely non-adaptive manner and independently of the challenge accumulator Acc. That is, our results apply already for a seemingly much weaker notion of security.

In addition, note that the output of the setup algorithm consists of two values: A public value Acc (the accumulator itself) that is used by both the Prove algorithm and the Vrfy algorithm, and a private state state that is used only by the Prove algorithm (the private state may include, for example, the randomness that was used by the Setup algorithm, for generating the accumulator).

Finally, as standard in the generic-group model, the above definition restricts only the number of queries issued by the adversary, and does not restrict the adversary's internal computation (i.e., the definition considers computationally-unbounded adversaries). As a consequence, note that without loss of generality such an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ is deterministic, and there is no need to transfer any state information from $\mathcal{A}_0$ to $\mathcal{A}_1$ (this can at most double the number of queries issued by $\mathcal{A}$).

3 Our Lower Bound in the Generic-Group Model

In this section we prove our main technical result, providing a lower bound on the number of group-operation queries required for batch verification. We prove the following theorem.

Theorem 3.1. *Let $\mathcal{ACC}$ be an $(n_{\mathsf{Acc}}, \ell_{\mathsf{Acc}}, n_\pi, \ell_\pi, q)$-accumulator in the generic-group model over a domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$, for some polynomials $n_{\mathsf{Acc}} = n_{\mathsf{Acc}}(\lambda, k)$, $\ell_{\mathsf{Acc}} = \ell_{\mathsf{Acc}}(\lambda, k)$, $n_\pi = n_\pi(\lambda, k, t)$, $\ell_\pi = \ell_\pi(\lambda, k, t)$ and $q = q(\lambda, k, t)$, and let* OrderGen *be an order-generation algorithm. If $\mathcal{ACC}$ is secure with respect to* OrderGen*, then for any polynomials $k = k(\lambda) \geq 1$ and $t = t(\lambda) \leq k$ and for all sufficiently large $\lambda \in \mathbb{N}$ it holds that*

$$q(\lambda, t, k) = \Omega\left(t \cdot \frac{\log_2 \binom{|\mathcal{X}_\lambda|}{k} - \left[n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}} + 1) + \ell_{\mathsf{Acc}} \right]}{k} \cdot \frac{1}{\log \lambda} \right).$$

As discussed in Section 1.1, recall that $\log_2 \binom{|\mathcal{X}_\lambda|}{k}$ is the expected number of bits required for an exact representation of k elements, and that $n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}} +$

$1) + \ell_{\mathsf{Acc}}$ is the amount of information that is actually stored by a generic-group accumulator from its verification algorithm's point of view. The following corollary of Theorem 3.1 shows that as long as the amount of information stored by an accumulator is bounded away from the information-theoretic amount that is required for an exact representation, then non-trivial batch verification is impossible.

Corollary 3.2. *Let $\mathcal{ACC}$ be an $(n_{\mathsf{Acc}}, \ell_{\mathsf{Acc}}, n_\pi, \ell_\pi, q)$-accumulator in the generic-group model over a domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$, for some polynomials $n_{\mathsf{Acc}} = n_{\mathsf{Acc}}(\lambda, k)$, $\ell_{\mathsf{Acc}} = \ell_{\mathsf{Acc}}(\lambda, k)$, $n_\pi = n_\pi(\lambda, k, t)$, $\ell_\pi = \ell_\pi(\lambda, k, t)$ and $q = q(\lambda, k, t)$, and let* $\mathsf{OrderGen}$ *be an order-generation algorithm. If $\mathcal{ACC}$ is secure with respect to* $\mathsf{OrderGen}$ *and $|\mathcal{X}_\lambda| = 2^{\Omega(\lambda)}$, then for any polynomials $k = k(\lambda) \geq 1$ and $t = t(\lambda) \leq k$, for any $0 < \epsilon < 1$ and for all sufficiently large $\lambda \in \mathbb{N}$, either*

$$n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}} + 1) + \ell_{\mathsf{Acc}} \geq (1 - \epsilon) \cdot \log_2 \binom{|\mathcal{X}_\lambda|}{k}$$

or

$$q(\lambda, k, t) = \Omega\left(t \cdot \frac{\epsilon\lambda}{\log \lambda}\right).$$

We prove the following lemma from which we then derive Theorem 3.1 and Corollary 3.2.

Lemma 3.3. *Let $\mathcal{ACC}$ be an $(n_{\mathsf{Acc}}, \ell_{\mathsf{Acc}}, n_\pi, \ell_\pi, q)$-accumulator in the generic-group model over a domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$, for some polynomials $n_{\mathsf{Acc}} = n_{\mathsf{Acc}}(\lambda, k)$, $\ell_{\mathsf{Acc}} = \ell_{\mathsf{Acc}}(\lambda, k)$, $n_\pi = n_\pi(\lambda, k, t)$, $\ell_\pi = \ell_\pi(\lambda, k, t)$ and $q = q(\lambda, k, t)$, and let* $\mathsf{OrderGen}$ *be an order-generation algorithm. If $\mathcal{ACC}$ is secure with respect to* $\mathsf{OrderGen}$ *then for any polynomials $k = k(\lambda) \geq 1$ and $t = t(\lambda) \leq k$ and for all sufficiently large $\lambda \in \mathbb{N}$ it holds that*

$$\frac{1}{2} \cdot \binom{|\mathcal{X}_\lambda|}{k} < (n_{\mathsf{Acc}} + 1)^{n_{\mathsf{Acc}}} \cdot 2^{\ell_{\mathsf{Acc}}} \cdot (n_{\mathsf{Acc}} + n_\pi + 3q + 1)^{6q \cdot \lceil k/t \rceil} \qquad (1)$$

In what follows, in Section 3.1 we prove Lemma 3.3, and then in Section 3.2 we rely on Lemma 3.3 for deriving the proofs of Theorem 3.1 and Corollary 3.2.

3.1 Proof of Lemma 3.3

For simplicity, we first prove the lemma for the case of known-order groups, and then show that the proof extends to unknown-order groups. The proof of Lemma 3.3 relies on the following notation given an $(n_{\mathsf{Acc}}, \ell_{\mathsf{Acc}}, n_\pi, \ell_\pi, q)$-generic-group accumulator $\mathcal{ACC} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy})$ over a domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ (recall Definition 2.1):

– In any execution of the verification algorithm $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S, \pi)$, note that the table $\mathbf{B}$ which stores group elements (and to which the oracle $\mathcal{O}$ provide black-box access – as described in Section 2.1) consists of at most $n_{\mathsf{Acc}} + n_\pi + q + 1$

entries: The table contains the generator $1 \in \mathbb{Z}_N$ in its first entry (as standard for all computations in this model), then it contains the n_{Acc} group elements that are part of the accumulator Acc, the n_π group elements that are part of the proof π, and finally at most q additional group elements that result from the group-operation queries issued by the verification algorithm. In addition, recall that each such query can be specified by providing a pair of indices to entries in the table together with the query type (i.e., the group operation $+$ or the group operation $-$). Therefore, each query has at most $2\left(n_{\mathsf{Acc}} + n_\pi + q + 1\right)^2$ possibilities. We let $\mathsf{VrfyQueries}_{\mathsf{Acc},S,\pi}$ denote the concatenation of the encodings of all queries made during the computation $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S, \pi)$ (in the order in which the queries were issued). Thus, the total number of possibilities for $\mathsf{VrfyQueries}_{\mathsf{Acc},S,\pi}$ is at most

$$\left(2 \cdot (n_{\mathsf{Acc}} + n_\pi + q + 1)^2\right)^q \leq (n_{\mathsf{Acc}} + n_\pi + q + 1)^{3q},$$

since $2 \leq n_{\mathsf{Acc}} + n_\pi + q + 1$.

– In any execution of the verification algorithm $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S, \pi)$ we would also like to encode the equality pattern of all group elements in the table $\mathbf{B}$. Recall that the table contains the generator $1 \in \mathbb{Z}_N$, the n_{Acc} group elements that are part of the accumulator Acc, the n_π group elements that are part of the proof π, and then at most q additional group elements that result from the group-operation queries issued by the verification algorithm. We split this encoding into the following three ingredients:
  - The equality pattern for the generator $1 \in \mathbb{Z}_N$ and the n_{Acc} group elements that are part of the accumulator Acc (i.e., for the $n_{\mathsf{Acc}} + 1$ first entries of the table) can be encoded as follows: For each of the n_{Acc} group elements that are part of the accumulator Acc we encode the index of the minimal entry among the first $n_{\mathsf{Acc}} + 1$ entries of the table that contains the same group element (independently of whether a corresponding equality query was explicitly issued by the verification algorithm). We denote this encoding by $\mathsf{AccEqualities}_{\mathsf{Acc}}$. There are at most $(n_{\mathsf{Acc}} + 1)^{n_{\mathsf{Acc}}}$ possibilities for $\mathsf{AccEqualities}_{\mathsf{Acc}}$.
  - The equality pattern for the n_π group elements that are part of the proof π (i.e., for the next n_π entries of the table) can be similarly encoded which results in at most $(n_{\mathsf{Acc}} + n_\pi + 1)^{n_\pi}$ possibilities. However, n_π can be significantly larger than q, and this may potentially lead to a too-long encoding for the purpose of our proof.
    Thus, instead of encoding the equality pattern among all n_π group elements that are part of the proof π, it is in fact sufficient for us to encode the equality pattern only among those elements that are involved in the group-operation queries that are issued during the computation $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S, \pi)$. There are at most q such queries, and therefore we need to encode the equality pattern only among at most $2q$ elements out of the n_π group elements that are part of the proof π. For each such element we encode the index of the minimal entry among the first $n_{\mathsf{Acc}} + 2q + 1$ entries of the table that contains the same group element (not

including the entries that are not involved in any of the group-operation queries). The number of possibilities for $\mathsf{ProofEqualities}_{\mathsf{Acc},S,\pi}$, is at most $(n_{\mathsf{Acc}} + 2q + 1)^{2q}$.

- The equality pattern for the (at most) q group elements that result from the group-operation queries issued by the verification algorithm (i.e., for the last q entries of the table) can be encoded the same way (while again not including the entries of the proof π that are not involved in any of the group-operation queries) resulting in at most $(n_{\mathsf{Acc}} + 2q + q + 1)^q$ possibilities. We denote this encoding by $\mathsf{QueriesEqualities}_{\mathsf{Acc},S,\pi}$.

Equipped with the above notation, we now prove Lemma 3.3.

Proof of Lemma 3.3. Let $\mathcal{ACC} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy})$ be an $(n_{\mathsf{Acc}}, \ell_{\mathsf{Acc}}, n_\pi, \ell_\pi, q)$-generic-group accumulator for some $n_{\mathsf{Acc}} = n_{\mathsf{Acc}}(\lambda, k)$, $\ell_{\mathsf{Acc}} = \ell_{\mathsf{Acc}}(\lambda, k)$, $n_\pi = n_\pi(\lambda, k, t)$, $\ell_\pi = \ell_\pi(\lambda, k, t)$ and $q = q(\lambda, k, t)$, and let $\mathsf{OrderGen}$ be an order-generation algorithm. Fix any polynomials $k = k(\lambda) \geq 1$ and $t = t(\lambda) \leq k$. We show that if Eq. (1) does not hold for infinitely many values of $\lambda \in \mathbb{N}$, then there exists a generic-group attacker $\mathcal{A}$ that issues a polynomial number of queries for which $\Pr\left[\mathsf{Expt}_{\mathcal{ACC},\mathcal{A}}(\lambda) = 1\right]$ is non-negligible in the security parameter $\lambda \in \mathbb{N}$ (recall that the experiment $\mathsf{Expt}_{\mathcal{ACC},\mathcal{A}}(\lambda)$ was defined in Definition 2.3 for capturing the security of generic-group accumulators).

At a high level, for any security parameter $\lambda \in \mathbb{N}$ our attacker $\mathcal{A}$, participating in the experiment $\mathsf{Expt}_{\mathcal{ACC},\mathcal{A}}(\lambda)$, will choose a random set $X \subseteq \mathcal{X}_\lambda$ of k elements for which the setup algorithm Setup will honestly generate an accumulator. Then, $\mathcal{A}$ will partition S into subsets of size t, and ask for an honestly-generated batch membership proof for each such subset. Then, with high probability, this will allow $\mathcal{A}$ to forge a batch membership proof for a set $S^* \nsubseteq X$ of size t.

In what follows we first describe the attacker $\mathcal{A}$ and then analyze its success probability. For simplicity we assume throughout the proof that t divide k, and we let $v = k/t$ (this is not essential and can be trivially avoided at the cost of somewhat degrading the readability of the proof). In addition, we let $<$ denote any ordering of the elements of the set $\mathcal{X} = \{\mathcal{X}_\lambda\}_{n \in \mathbb{N}}$ (e.g., the lexicographic order). As discussed in Section 2.1, recall that for a group element g and for a vector of group elements v, we will differentiate between the case where g and v are provided explicitly and the case where they are provided implicitly via the table $\mathbf{B}$, using the notation g and v in the former case, and the notation $\widehat{g}$ and $\widehat{v}$ in the latter.

The attacker $\boldsymbol{\mathcal{A}} = (\boldsymbol{\mathcal{A}_0}, \boldsymbol{\mathcal{A}_1})$

The algorithm $\boldsymbol{\mathcal{A}_0}$. On input $(1^\lambda, N)$ and oracle access to $\mathcal{O}(\cdot)$, the algorithm $\mathcal{A}_0$ samples a uniformly distributed set $X \subseteq \mathcal{X}_\lambda$ that consists of k distinct elements $x_1 < \cdots < x_k$. It then outputs the set X, and also passes it as its internal state to the algorithm $\mathcal{A}_1$.

The algorithm $\boldsymbol{\mathcal{A}_1}$. On input $(1^\lambda, N, \mathsf{Acc}, X)$ and oracle access to $\mathcal{O}(\cdot)$ and to $\mathsf{Prove}^{\mathcal{O}}(\mathsf{Acc}, \mathsf{state}, \cdot)$, where $(\mathsf{Acc}, \mathsf{state}) \leftarrow \mathsf{Setup}^{\mathcal{O}}(1^\lambda, N, X)$ is honestly-generated within the experiment $\mathsf{Expt}_{\mathcal{ACC},\mathcal{A}}(\lambda)$, the algorithm $\mathcal{A}_1$ is defined as follows:

1. The algorithm $\mathcal{A}_1$ computes the equality pattern $\mathsf{AccEqualities}_{\mathsf{Acc}}$ by issuing equality queries (recall that $\mathsf{Acc} = (\widehat{\mathsf{Acc}_{\mathsf{G}}}, \mathsf{Acc}_{\mathsf{str}})$, where $\mathsf{Acc}_{\mathsf{G}}$ is a sequence of $n_{\mathsf{Acc}}(\lambda, k)$ group elements that can be accessed indirectly via oracle queries, and $\mathsf{Acc}_{\mathsf{str}} \in \{0,1\}^{\ell_{\mathsf{Acc}}(\lambda,k)}$ is an explicit string that can be accessed directly).
2. For every $i \in [v]$ the algorithm $\mathcal{A}_1$ queries the oracle $\mathsf{Prove}^{\mathcal{O}}(\mathsf{Acc}, \mathsf{state}, \cdot)$ with the set $S_i = \{x_{(i-1)\cdot t+1}, \ldots, x_{i\cdot t}\}$ to obtain a proof $\pi_i \leftarrow \mathsf{Prove}^{\mathcal{O}}(\mathsf{Acc}, \mathsf{state}, S_i)$. We denote $\pi_i = (\widehat{\pi_{i,G}}, \pi_{i,\mathsf{str}})$, where $\pi_{i,G}$ is a sequence of $n_\pi(\lambda, t, k)$ group elements that can be accessed indirectly via oracle queries and $\pi_{i,\mathsf{str}} \in \{0,1\}^{\ell_\pi(\lambda,t,k)}$ is an explicit string that can be accessed directly.
   Then, the algorithm $\mathcal{A}_1$ executes $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_i, \pi_i)$ for obtaining the query pattern $\mathsf{VrfyQueries}_{\mathsf{Acc},S_i,\pi_i}$ by forwarding the queries issued by Vrfy to the oracle $\mathcal{O}$, and issues additional equality queries for computing the equality patterns $\mathsf{ProofEqualities}_{\mathsf{Acc},S_i,\pi_i}$ and $\mathsf{QueriesEqualities}_{\mathsf{Acc},S_i,\pi_i}$.
3. The algorithm $\mathcal{A}_1$ finds a set $X' \subseteq \mathcal{X}_\lambda$ that consists of k distinct elements $x'_1 < \cdots < x'_k$, and strings $r', r'_1, \ldots, r'_v \in \{0,1\}^*$ satisfying the following requirements:
   - $X' \neq X$.
   - $\mathsf{AccEqualities}_{\mathsf{Acc}'} = \mathsf{AccEqualities}_{\mathsf{Acc}}$ and $\mathsf{Acc}'_{\mathsf{str}} = \mathsf{Acc}_{\mathsf{str}}$, where $(\mathsf{Acc}', \mathsf{state}') = \mathsf{Setup}(1^\lambda, N, X'; r')$ and $\mathsf{Acc}' = (\mathsf{Acc}'_{\mathsf{G}}, \mathsf{Acc}_{\mathsf{str}})$. Note that all inputs to the computation $\mathsf{Setup}(1^\lambda, N, X'; r')$ are explicitly known to $\mathcal{A}_1$, and therefore this computation can be internally emulated without any oracle queries.
   - For every $i \in [v]$ it holds that
   $$\begin{aligned} \mathsf{VrfyQueries}_{\mathsf{Acc}',S'_i,\pi'_i} &= \mathsf{VrfyQueries}_{\mathsf{Acc},S_i,\pi_i} \\ \mathsf{ProofEqualities}_{\mathsf{Acc}',S'_i,\pi'_i} &= \mathsf{ProofEqualities}_{\mathsf{Acc},S_i,\pi_i} \\ \mathsf{QueriesEqualities}_{\mathsf{Acc}',S'_i,\pi'_i} &= \mathsf{QueriesEqualities}_{\mathsf{Acc},S_i,\pi_i} \end{aligned}$$
     where $\pi'_i = \mathsf{Prove}(\mathsf{Acc}', \mathsf{state}', S'_i; r'_i)$ and $S'_i = \{x'_{(i-1)\cdot t+1}, \ldots, x'_{i\cdot t}\}$.

   If such a set X' and strings $r', r'_1, \ldots, r'_v \in \{0,1\}^*$ do not exist, then the algorithm $\mathcal{A}_1$ aborts the experiment.
4. Let $i^* \in [v]$ be any index such that $S'_{i^*} \not\subseteq X$ (e.g., the smallest one), then the algorithm $\mathcal{A}_1$ outputs $S^* = S'_{i^*}$ and $\pi^* = \left(\widehat{\pi^*_G}, \pi^*_{\mathsf{str}}\right)$, where π^*_G is a sequence of n_π group elements that are defined below and $\pi^*_{\mathsf{str}} = \pi'_{i^*,\mathsf{str}}$ is an explicit string.
   (a) Let $\mathcal{J} \subseteq [n_\pi]$ be the positions of the group elements that are part of the proof π_{i^*} which are accessed by the group-operation queries issued during the computation $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi_{i^*})$.
   (b) For every $j \in \mathcal{J}$ we define $(\pi^*_G)_j = (\pi_{i^*,G})_j$ (i.e., we set π^*_G to agree with $\pi_{i^*,G}$ on the group elements in the positions included in $\mathcal{J}$).
   (c) Let $T = 1 + n_{\mathsf{Acc}} + n_\pi + q$, and for every $j \in [T]$ we denote by V_j and V'_j the group element at the jth entry of the table $\mathbf{B}$ in the computations $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi_{i^*})$ and $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}', S'_{i^*}, \pi'_{i^*})$, respectively. Note that $T = 1 + n_{\mathsf{Acc}} + n_\pi + q$ is indeed an upper bound on the number of entries in the table $\mathbf{B}$ in these computations: The first entry contains the element $1 \in \mathbb{Z}_N$, the next n_{Acc} entries contain the group elements of the given accumulator, the next n_π entries contains the group elements of the given proof, and then there are at most q entries that result from the

group-operation queries issued by the verification algorithm. Let $\mathcal{I} = \{1, \ldots, 1+n_{\mathsf{Acc}}\} \cup (1+n_{\mathsf{Acc}}+\mathcal{J}) \cup \{1+n_{\mathsf{Acc}}+n_\pi+1, \ldots, 1+n_{\mathsf{Acc}}+n_\pi+q\} \subseteq [T]$.
[Recall that $1 + n_{\mathsf{Acc}} + \mathcal{J} = \{1 + n_{\mathsf{Acc}} + j | j \in \mathcal{J}\}$.]

(d) For every $j \in [n_\pi] \setminus \mathcal{J}$ in increasing order we define $(\pi^*_G)_j$ as follows:
   i. If there exists a position $w \in \mathcal{I}$ such that $(\pi'_{i^*,G})_j = V'_w$, then we define $(\pi^*_G)_j = V_w$.
   ii. Otherwise, if for all positions $w \in \mathcal{I}$ it holds that $(\pi'_{i^*,G})_j \neq V'_w$ then
      A. If there exists some $k \in [n_\pi] \setminus \mathcal{J}$ such that $k < j$ and $(\pi'_{i^*,G})_j = (\pi'_{i^*,G})_k$, then define $(\pi^*_G)_j = (\pi^*_G)_k$ (note that $(\pi^*_G)_k$ is already defined in this stage since $k < j$).
      B. Otherwise, we define $(\pi^*_G)_j$ arbitrarily such that $(\pi^*_G)_j \neq V_w$ for all $w \in \mathcal{I}$ and $(\pi^*_G)_j \neq (\pi^*_G)_k$ for all $k \in [n_\pi] \setminus \mathcal{J}$ such that $k < j$.

At this point, after having described our attacker $\mathcal{A}$, we are ready to analyze its success probability: In Claim 3.4 we prove that $\mathcal{A}$ aborts with probability at most 1/2, and in Claim 3.5 we prove that any execution in which $\mathcal{A}$ does not abort results in a successful forgery. First, however, we observe that the query complexity of our attacker is polynomial in $k(\lambda)$, $n_{\mathsf{Acc}}(\lambda, k)$, $n_\pi(\lambda, k, t)$ and $q(\lambda, k, t)$, and thus polynomial in the security parameter $\lambda \in \mathbb{N}$. Specifically, the algorithm $\mathcal{A}_0$ does not issue any queries, and the algorithm $\mathcal{A}_1$ issues the following queries:

– Step 1: This step requires at most $(n_{\mathsf{Acc}}(\lambda, k))^2$ queries for computing the equality pattern $\mathsf{AccEqualities}_{\mathsf{Acc}}$ among the group elements Acc_G of the given accumulator Acc.
– Step 2: This step requires v queries for obtaining the proofs $\pi_1, \ldots, \pi_v$, and at most $v \cdot (n_\pi(\lambda, t, k) + n_{\mathsf{Acc}}(\lambda, k))^2$ queries for computing the equality patterns $\mathsf{ProofEqualities}_{\mathsf{Acc}, S_i, \pi_i}$ among the group elements $\pi_{i,G}$ of the proofs $\pi_1, \ldots, \pi_v$.
  In addition, this step requires at most $v \cdot q(\lambda, t, k) + v \cdot (q(\lambda, t, k) + n_\pi(\lambda, t, k) + n_{\mathsf{Acc}}(\lambda, k))^2$ queries for computing the query patterns $\mathsf{VrfyQueries}_{\mathsf{Acc}, S_1, \pi_1}, \ldots, \mathsf{VrfyQueries}_{\mathsf{Acc}, S_v, \pi_v}$ and the query equality patterns $\mathsf{QueriesEqualities}_{\mathsf{Acc}, S_1, \pi_1}, \ldots, \mathsf{QueriesEqualities}_{\mathsf{Acc}, S_v, \pi_v}$.
– Step 3: No queries. All inputs to the relevant computations are explicitly known to $\mathcal{A}_1$, and therefore these computations can be internally emulated without any oracle queries.
– Step 4: The sub-steps 4(a) – 4(c) do not require any queries, whereas sub-step 4(d) does require issuing both group-operation queries and equality queries. Specifically, in sub-step 4(d).ii.B. the attacker defines $(\pi^*_G)_j$ arbitrarily such that $(\pi^*_G)_j \neq V_w$ for all $w \in \mathcal{I}$ and $(\pi^*_G)_j \neq (\pi^*_G)_k$ for all $k \in [n_\pi] \setminus \mathcal{J}$ such that $k < j$. This can be done, for example, by adding $1 \in \mathbb{Z}_N$ to $(\pi^*_G)_j$ in an iterative manner until $(\pi^*_G)_j \neq V_w$ for all $w \in \mathcal{I}$ and $(\pi^*_G)_j \neq (\pi^*_G)_k$ for all $k \in [n_\pi] \setminus \mathcal{J}$ such that $k < j$. The number of such iterations is upper bounded by the number of distinct elements in the table $\mathbf{B}$, which is at most $1 + n_{\mathsf{Acc}}(\lambda, k) + n_\pi(\lambda, t, k) + q(\lambda, t, k)$ (the number of entries in $\mathbf{B}$).

Claim 3.4 *For any* $\lambda \in \mathbb{N}$, *if*

$$\frac{1}{2} \cdot \binom{|\mathcal{X}_\lambda|}{k} \geq (n_{\mathsf{Acc}} + 1)^{n_{\mathsf{Acc}}} \cdot 2^{\ell_{\mathsf{Acc}}} \cdot (n_{\mathsf{Acc}} + n_\pi + 3q + 1)^{6q \cdot \lceil k/t \rceil} \tag{2}$$

then $\Pr[\mathcal{A} \text{ aborts}] < 1/2$.

Proof of Claim 3.4. We show that if Eq. (2) holds then with probability at least $1/2$ the attacker is able to find a set $X' \subseteq \mathcal{X}_\lambda$ that consists of k distinct elements $x'_1 < \cdots < x'_k$, and strings $\vec{r'} = (r', r'_1, \ldots, r'_v) \in \{0,1\}^*$, that satisfy the requirements specified in Step 3. Denote by $r \in \{0,1\}^*$ the randomness used by the algorithm Setup in the experiment $\mathsf{Expt}_{\mathcal{ACC},\mathcal{A}}(\lambda)$ (i.e., $(\mathsf{Acc}, \mathsf{state}) = \mathsf{Setup}^{\mathcal{O}}(1^\lambda, N, X; r)$). In addition, for every $i \in [v]$ denote by $r_i \in \{0,1\}^*$ the randomness used by the oracle $\mathsf{Prove}^{\mathcal{O}}(\mathsf{Acc}, \mathsf{state}, \cdot)$ when computing a batch membership proof for the set S_i in the experiment $\mathsf{Expt}_{\mathcal{ACC},\mathcal{A}}(\lambda)$ (i.e., $\pi_i = \mathsf{Prove}^{\mathcal{O}}(\mathsf{Acc}, \mathsf{state}, S_i; r_i)$), and let $\vec{r} = (r, r_1, \ldots, r_v)$. We show that even when restricting the attacker to choose $\vec{r'} = \vec{r}$ there is still a set X' that satisfies the requirements specified in Step 3 with probability at least $1/2$ over the choice of X.

Consider the function $F_{\vec{r}}$ that takes as input a set $X \subseteq \mathcal{X}_\lambda$ of k distinct elements $x_1 < \cdots < x_k$, and returns as output the following values:

$$\begin{aligned} F_{\vec{r}}(X) = \Big(&\mathsf{AccEqualities}_{\mathsf{Acc}}, \mathsf{Acc}_{\mathsf{str}}, \\ &\mathsf{VrfyQueries}_{\mathsf{Acc},S_1,\pi_1}, \ldots, \mathsf{VrfyQueries}_{\mathsf{Acc},S_v,\pi_v}, \\ &\mathsf{ProofEqualities}_{\mathsf{Acc},S_1,\pi_1}, \ldots, \mathsf{ProofEqualities}_{\mathsf{Acc},S_v,\pi_v}, \\ &\mathsf{QueriesEqualities}_{\mathsf{Acc},S_1,\pi_1} \ldots, \mathsf{QueriesEqualities}_{\mathsf{Acc},S_v,\pi_v}\Big) \end{aligned}$$

where $S_i = \{x_{(i-1)\cdot t+1}, \ldots, x_{i \cdot t}\}$ for every $i \in [v]$, $(\mathsf{Acc}, \mathsf{sk}) \leftarrow \mathsf{Setup}^{\mathcal{O}}(\lambda, N, X; r)$, and $\pi_i \leftarrow \mathsf{Prove}(\mathsf{Acc}, \mathsf{sk}, S_i; r_i)$ for every $i \in [v]$. Our goal is to prove that with probability at least $1/2$ over the choice of X there exists a set $X' \neq X$ such that $F_{\vec{r}}(X') = F_{\vec{r}}(X)$. We prove this claim by showing that the size of the image of the function $F_{\vec{r}}$, denoted $\mathsf{Image}(F_{\vec{r}})$, is at most half the size of its domain (this guarantees that with probability at least $1/2$ over the choice of X there exists a set $X' \neq X$ as required).

The domain of the function $F_{\vec{r}}$ is of size $\binom{|\mathcal{X}_\lambda|}{k}$. The number of possibilities for an output of the function $F_{\vec{r}}$ is the product of the following quantities (as discussed above when defining $\mathsf{AccEqualities}_{\mathsf{Acc}}$, $\mathsf{VrfyQueries}_{\mathsf{Acc},S_i,\pi_i}$, $\mathsf{ProofEqualities}_{\mathsf{Acc},S_i,\pi_i}$ and $\mathsf{QueriesEqualities}_{\mathsf{Acc},S_i,\pi_i}$):

- $\mathsf{AccEqualities}_{\mathsf{Acc}}$ and $\mathsf{Acc}_{\mathsf{str}}$ have $(n_{\mathsf{Acc}} + 1)^{n_{\mathsf{Acc}}}$ and $2^{\ell_{\mathsf{Acc}}}$ possibilities, respectively.
- $\mathsf{VrfyQueries}_{\mathsf{Acc},S_i,\pi_i}$ for every $i \in [v]$ has $(n_{\mathsf{Acc}} + n_\pi + q + 1)^{3q}$ possibilities.
- $\mathsf{ProofEqualities}_{\mathsf{Acc},S_i,\pi_i}$ for every $i \in [v]$ has $(n_{\mathsf{Acc}} + 2q + 1)^{2q}$ possibilities.

– $\mathsf{QueriesEqualities}_{\mathsf{Acc},S_i,\pi_i}$ for every $i \in [v]$ has $(n_{\mathsf{Acc}} + 3q + 1)^q$ possibilities.

Thus, the size of the image of the function $F_{\vec{r}}$ can be upper bounded via

$$|\mathsf{Image}(F_{\vec{r}})| \leq (n_{\mathsf{Acc}} + 1)^{n_{\mathsf{Acc}}} \cdot 2^{\ell_{\mathsf{Acc}}} \cdot (n_{\mathsf{Acc}} + n_\pi + 3q + 1)^{6q \cdot \lceil k/t \rceil} .$$

We assume that Eq. (2) holds, and therefore the size of the image of the function $F_{\vec{r}}$ is at most half the size of its domain, and the claim follows.

■

Claim 3.5 *For any $\lambda \in \mathbb{N}$ it holds that*

$$\Pr\left[\mathsf{Expt}_{\mathcal{ACC},\mathcal{A}}(\lambda) = 1 \middle| \mathcal{A} \text{ does not abort}\right] = 1.$$

Proof of Claim 3.5. Assuming that $\mathcal{A}$ does not abort we prove that $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S'_{i^*}, \pi^*) = 1$. Together with the fact that $S'_{i^*} \not\subseteq X$, this implies that $\mathsf{Expt}_{\mathcal{ACC},\mathcal{A}}(\lambda) = 1$. Recall that the proof $\pi^* = \left(\widehat{\pi^*_G}, \pi^*_{\mathsf{str}}\right)$ is constructed using the two proofs π_{i^*} and π'_{i^*}, where:

– $\mathcal{A}$ queried the oracle $\mathsf{Prove}^{\mathcal{O}}(\mathsf{Acc}, \mathsf{state}, \cdot)$ with the set S_{i^*} to obtain $\pi_{i^*} = \left(\widehat{\pi_{i^*,G}}, \pi_{i^*,\mathsf{str}}\right) \leftarrow \mathsf{Prove}^{\mathcal{O}}(\mathsf{Acc}, \mathsf{state}, S_{i^*})$, where $\pi_{i^*,G}$ is a sequence of group elements and $\pi_{i^*,\mathsf{str}}$ is an explicit string.

– $\mathcal{A}$ generated $\pi'_{i^*} = \left(\widehat{\pi'_{i^*,G}}, \pi'_{i^*,\mathsf{str}}\right) \leftarrow \mathsf{Prove}(\mathsf{Acc}', \mathsf{state}', S'_i; r'_i)$ subject to the requirements specified in the description of the attack, where $\pi'_{i^*,G}$ is a sequence of group elements and $\pi'_{i^*,\mathsf{str}}$ is an explicit string.

The correctness of the accumulator guarantees that $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi_{i^*}) = 1$ and $\mathsf{Vrfy}(\mathsf{Acc}', S'_{i^*}, \pi'_{i^*}) = 1$, and we show that $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi^*) = 1$. This will follow from the fact that the computation of the verification algorithm, which can access group elements only via the oracle $\mathcal{O}$, cannot distinguish between the two inputs $(\mathsf{Acc}', S'_{i^*}, \pi'_{i^*})$ and $(\mathsf{Acc}, S'_{i^*}, \pi^*)$.

Recall that each computation is associated with a table $\mathbf{B}$, where each entry of this table stores an element of $\mathbb{Z}_N$, and that the oracle $\mathcal{O}$ provides black-box access to $\mathbf{B}$ via group operations and equality queries. Let $T = 1 + n_{\mathsf{Acc}} + n_\pi + q$, and for every $j \in [T]$ we denote by V_j, V'_j and V^*_j the $\mathbb{Z}_N$ element that is located at the jth entry of the table $\mathbf{B}$ in the computations $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi_{i^*})$, $\mathsf{Vrfy}(\mathsf{Acc}', S'_{i^*}, \pi'_{i^*})$ and $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S'_{i^*}, \pi^*)$, respectively.

Recall that we denoted by $\mathcal{J} \subseteq [n_\pi]$ the positions of the group elements that are part of the proof π_{i^*} which are accessed by the group-operation queries issued during the computation $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi_{i^*})$. Recall also that we defined $\mathcal{I} = \{1, \ldots, 1+n_{\mathsf{Acc}}\} \cup (1+n_{\mathsf{Acc}}+\mathcal{J}) \cup \{1+n_{\mathsf{Acc}}+n_\pi+1, \ldots, 1+n_{\mathsf{Acc}}+n_\pi+q\} \subseteq [T]$. Observe that $(V_1, \ldots, V_T)$ and $(V'_1, \ldots, V'_T)$, have the same equality pattern when restricted to the position included in $\mathcal{I}$ (although they may correspond to different $\mathbb{Z}_N$ elements), since based on the description of our attacker it holds that

$$\mathsf{AccEqualities}_{\mathsf{Acc}'} = \mathsf{AccEqualities}_{\mathsf{Acc}}$$
$$\mathsf{ProofEqualities}_{\mathsf{Acc}',S'_{i^*},\pi'_{i^*}} = \mathsf{ProofEqualities}_{\mathsf{Acc},S_{i^*},\pi_{i^*}}$$
$$\mathsf{QueriesEqualities}_{\mathsf{Acc}',S'_{i^*},\pi'_{i^*}} = \mathsf{QueriesEqualities}_{\mathsf{Acc},S_{i^*},\pi_{i^*}} .$$

In addition, the same queries are issued in the computations $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi_{i^*})$ and $\mathsf{Vrfy}(\mathsf{Acc}', S'_{i^*}, \pi'_{i^*})$, as $\mathsf{VrfyQueries}_{\mathsf{Acc}', S'_{i^*}, \pi'_{i^*}} = \mathsf{VrfyQueries}_{\mathsf{Acc}, S_{i^*}, \pi_{i^*}}$.

Recall that for every $j \in \mathcal{J}$ we defined $(\pi^*_G)_j = (\pi_{i^*,G})_j$. Note that the first $1 + n_{\mathsf{Acc}} + n_\pi$ entries of the table $\mathbf{B}$ are the $\mathbb{Z}_N$ elements corresponding to the group elements that are provided as part of the inputs to the computation. In both the computations $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi_{i^*})$ and $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S'_{i^*}, \pi^*)$ the first $1 + n_{\mathsf{Acc}} + n_\pi$ entries of the table $\mathbf{B}$ are the elements $(1, \mathsf{Acc}_G, \pi_{i^*,G})$ and $(1, \mathsf{Acc}_G, \pi^*_G)$ respectively. Therefore, for every $w \in \mathcal{I} \cap [1 + n_{\mathsf{Acc}} + n_\pi]$ it holds that $V^*_w = V_w$. Since $(V_1, \ldots, V_T)$ and $(V'_1, \ldots, V'_T)$, have the same equality pattern on the indices in $\mathcal{I}$, we get that $(V'_1, \ldots, V'_{1+n_{\mathsf{Acc}}+n_\pi})$ and $(V^*_1, \ldots, V^*_{1+n_{\mathsf{Acc}}+n_\pi})$ have the same equality pattern on the indices in $\mathcal{I}$. Recall also that for every $j \in [n_\pi] \setminus \mathcal{J}$ we defined $(\pi^*_G)_j$ according to the equalities in $(V'_1, ..., V'_T)$ using the elements in $(V_1, ..., V_T)$ or new element when needed. So $(V^*_1, \ldots, V^*_{1+n_{\mathsf{Acc}}+n_\pi})$ and $(V'_1, \ldots, V'_{1+n_{\mathsf{Acc}}+n_\pi})$ have the same equality pattern everywhere (i.e., not only when restricted to the positions included in $\mathcal{I}$).

In what follows we prove that $(V^*_1, \ldots, V^*_T)$ and $(V'_1, \ldots, V'_T)$ have the same equality pattern everywhere (although they may correspond to different $\mathbb{Z}_N$ elements). Together with the fact that the explicit inputs to their respective computations, $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S'_{i^*}, \pi^*)$ and $\mathsf{Vrfy}(\mathsf{Acc}', S'_{i^*}, \pi'_{i^*})$, are the same (these are the explicit bit-strings $\mathsf{Acc}_{\mathsf{str}}, S'_{i^*}$ and $\pi'_{i^*,\mathsf{str}}$), we obtain that $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S'_{i^*}, \pi^*) = \mathsf{Vrfy}(\mathsf{Acc}', S'_{i^*}, \pi'_{i^*})$ as required.

We prove, via induction on $j \in \{0, \ldots, q\}$, that (1) for every $w \in \mathcal{I} \cap [1 + n_{\mathsf{Acc}} + n_\pi + j]$ it holds that $V^*_w = V_w$, and (2) $(V^*_1, \ldots, V^*_{1+n_{\mathsf{Acc}}+n_\pi+j})$ and $(V'_1, \ldots, V'_{1+n_{\mathsf{Acc}}+n_\pi+j})$ have the same equality pattern. For the case $j = 0$ this has already been established above.

Now assume that for some $j \in \{0, \ldots, q-1\}$ we have that for every $w \in \mathcal{I} \cap [1 + n_{\mathsf{Acc}} + n_\pi + j]$ it holds that $V^*_w = V_w$, and that $(V^*_1, \ldots, V^*_{1+n_{\mathsf{Acc}}+n_\pi+j})$ and $(V'_1, \ldots, V'_{1+n_{\mathsf{Acc}}+n_\pi+j})$ have the same equality pattern. We would like to argue that the same holds for $j + 1$ as well. The entries $V^*_{1+n_{\mathsf{Acc}}+n_\pi+j+1}$ and $V_{1+n_{\mathsf{Acc}}+n_\pi+j+1}$ contain the result of the next group-operation query in the computations $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S'_{i^*}, \pi^*)$ and $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi_{i^*})$. The next group-operation query in the computation $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S'_{i^*}, \pi^*)$ is identical to that of the computation $\mathsf{Vrfy}(\mathsf{Acc}', S'_{i^*}, \pi'_{i^*})$ (since both computations have the same explicit inputs and so far have the same equality patterns in their tables), and the next group-operation query in the computation $\mathsf{Vrfy}(\mathsf{Acc}', S'_{i^*}, \pi'_{i^*})$ is identical to that of the computation $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi_{i^*})$ (since we required $\mathsf{VrfyQueries}_{\mathsf{Acc}', S'_{i^*}, \pi'_{i^*}} = \mathsf{VrfyQueries}_{\mathsf{Acc}, S_{i^*}, \pi_{i^*}}$). Therefore, the next group-operation query in the computation $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S'_{i^*}, \pi^*)$ is identical to that of the computation $\mathsf{Vrfy}^{\mathcal{O}}(\mathsf{Acc}, S_{i^*}, \pi_{i^*})$. Since the two tables $(V^*_1, \ldots, V^*_{1+n_{\mathsf{Acc}}+n_\pi+j})$ and $(V_1, \ldots, V_{1+n_{\mathsf{Acc}}+n_\pi+j})$ are identical on the indices in $\mathcal{I}$, which contain the indices of the queries of Vrfy, this implies that $(V^*_1, \ldots, V^*_{1+n_{\mathsf{Acc}}+n_\pi+j+1})$ and $(V_1, \ldots, V_{1+n_{\mathsf{Acc}}+n_\pi+j+1})$ are identical on the indices in $\mathcal{I}$ (which proves part (1)).

Note that $(V_1, \ldots, V_{1+n_{\mathsf{Acc}}+n_\pi+j+1})$ and $(V'_1, \ldots, V'_{1+n_{\mathsf{Acc}}+n_\pi+j+1})$ have the same equality pattern on the indices in $\mathcal{I}$ (by the description of our attacker), and therefore $(V^*_1, \ldots, V^*_{1+n_{\mathsf{Acc}}+n_\pi+j+1})$ and $(V'_1, \ldots, V'_{1+n_{\mathsf{Acc}}+n_\pi+j+1})$ have the same equality pattern on the indices in $\mathcal{I}$. In addition, the elements $(\pi^*_G)_j$ of π^*_G for all $j \in [n_\pi] \setminus \mathcal{J}$ are chosen to agree with the equality pattern of $(V'_1, \ldots, V'_T)$. Thus, $(V^*_1, \ldots, V^*_{1+n_{\mathsf{Acc}}+n_\pi+j+1})$ and $(V'_1, \ldots, V'_{1+n_{\mathsf{Acc}}+n_\pi+j+1})$ have the same equality pattern. ■

This settles the proof of Lemma 3.3. ■

Extension to unknown-order groups. As discussed in Section 2.1, we consider two different flavors of generic groups: groups of known orders and groups of unknown orders. When modeling known-order generic groups then all algorithms receive the order of the underlying group as an explicit input, and when modeling unknown-order generic groups then the order is not provided (still, however, the corresponding order-generation algorithm $\mathsf{OrderGen}$ is publicly known). In our case, this difference corresponds to whether or not the accumulator's procedures Setup, Prove and Vrfy, and our attacker $\mathcal{A}$ receive the order of the group as input, and the above proof of Lemma 3.3 assumes that they do.

This proof easily extends to the case where the accumulator's procedures and our attacker do not receive the order of the group as input. Specifically, note that our attacker uses the order N of the group only in Step 3 of the algorithm $\mathcal{A}_1$, for finding a set $X' \subseteq \mathcal{X}_\lambda$ and randomness $\vec{r'} = (r', r'_1, \ldots, r'_v)$ that satisfy the prescribed requirements (finding these values requires $\mathcal{A}_1$ to internally perform computations modulo N). However, if the accumulator's procedures do not receive the order of the group as input, then we can modify the algorithm $\mathcal{A}_1$ to find in Step 3, together with X' and $\vec{r'}$, an integer N' in the support of the computation $\mathsf{OrderGen}(1^\lambda)$ such that the exact same conditions are satisfied (while performing the required internal computations modulo N').

The proof of Claim 3.4 is essentially unchanged, now showing that even when restricting the attacker to choose $\vec{r'} = \vec{r}$ and $N' = N$ there is still a set X' that satisfies the prescribed requirements with probability at least $1/2$ over the choice of X (i.e., there exists at least one suitable choice of $\vec{r'}$ and N' exactly as before). The proof of Claim 3.5 is completely unchanged, since the accumulator's procedures do not receive the order of the group as input, the exact same proof shows that the verification algorithm, which can access group elements only via the oracle $\mathcal{O}$, cannot distinguish between the two inputs $(\mathsf{Acc}', S'_{i^*}, \pi'_{i^*})$ and $(\mathsf{Acc}, S'_{i^*}, \pi^*)$.

3.2 Proofs of Theorem 3.1 and Corollary 3.2

Equipped with Lemma 3.3 we now derive Theorem 3.1 and Corollary 3.2.

Proof of Theorem 3.1. Lemma 3.3 implies that for all sufficiently large $\lambda \in \mathbb{N}$ it holds that

$$\begin{aligned}\log_2 \binom{|\mathcal{X}_\lambda|}{k} &< n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}}+1) + \ell_{\mathsf{Acc}} \\ &\quad + \left\lceil \frac{k}{t} \right\rceil \cdot 6q \cdot \log_2(n_{\mathsf{Acc}} + n_\pi + 3q + 1) + 1 \\ &\leq n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}}+1) + \ell_{\mathsf{Acc}} \\ &\quad + \frac{2k}{t} \cdot 6q \cdot \log_2(n_{\mathsf{Acc}} + n_\pi + 3q + 1) + 1.\end{aligned}$$

Therefore, using the fact that $t \leq k$ and $q \geq 1$ we obtain

$$\begin{aligned}t \cdot \frac{\log_2 \binom{|\mathcal{X}_\lambda|}{k} - \left[n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}}+1) + \ell_{\mathsf{Acc}}\right]}{k} &\leq 12q \log_2(n_{\mathsf{Acc}} + n_\pi + 3q + 1) + \frac{t}{k} \\ &\leq 12q \log_2(n_{\mathsf{Acc}} + n_\pi + 3q + 1) + 1 \\ &\leq 13q \log_2(n_{\mathsf{Acc}} + n_\pi + 3q + 1).\end{aligned}$$

Since the functions n_{Acc},n_π and q are all polynomials in the security parameter $\lambda \in \mathbb{N}$, then $\log_2(n_{\mathsf{Acc}} + n_\pi + 3q + 1) = O(\log_2 \lambda)$, and therefore

$$q = \Omega\left(t \cdot \frac{\log_2 \binom{|\mathcal{X}_\lambda|}{k} - \left[n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}}+1) + \ell_{\mathsf{Acc}}\right]}{k} \cdot \frac{1}{\log \lambda}\right).$$

■

Proof of Corollary 3.2. If we assume that

$$n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}}+1) + \ell_{\mathsf{Acc}} < (1-\epsilon) \cdot \log_2 \binom{|\mathcal{X}_\lambda|}{k}$$

then

$$\begin{aligned}\log_2 \binom{|\mathcal{X}_\lambda|}{k} - \left[n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}}+1) + \ell_{\mathsf{Acc}}\right] &> \epsilon \cdot \log_2 \binom{|\mathcal{X}_\lambda|}{k} \\ &\geq \epsilon k \cdot \log_2 \left(\frac{|\mathcal{X}_\lambda|}{k}\right) \\ &= \epsilon k \cdot \Omega(\lambda), \qquad (3)\end{aligned}$$

where Eq. (3) follows from the assumption that $|\mathcal{X}_\lambda| \geq 2^{\Omega(\lambda)}$ and the fact that $k = k(\lambda)$ is polynomial in the security parameter $\lambda \in \mathbb{N}$. Therefore,

$$\begin{aligned}q &= \Omega\left(t \cdot \frac{\log_2 \binom{|\mathcal{X}_\lambda|}{k} - \left[n_{\mathsf{Acc}} \cdot \log_2(n_{\mathsf{Acc}}+1) + \ell_{\mathsf{Acc}}\right]}{k} \cdot \frac{1}{\log \lambda}\right) \\ &= \Omega\left(t \cdot \frac{\epsilon\lambda}{\log \lambda}\right).\end{aligned}$$

■

4 Open Problems

In this section we briefly discuss several open problems that arise from this work.

Randomized verification and imperfect correctness. Our work considers accumulators with deterministic verification procedures and perfect correctness, as noted in Section 2.2. Although this seems to be the case with the known accumulators, the more general case of accumulators with randomized verification procedures and imperfect correctness (i.e., valid proofs are accepted with all but a negligible probability) is clearly fundamental, and thus an interesting direction for future research.

Non-trivial non-interactive batch verification in Shoup's model. As discussed in Section 1.2, we prove our result within the generic-group model introduced by Maurer [Mau05], which together with the incomparable model introduced by Shoup [Sho97], seem to be the most commonly used approaches for capturing generic-group computations. A natural open problem is whether our result can be either proved or circumvented within Shoup's model.

One should note that our result can be circumvented by applying the Fiat-Shamir transform [BBF19, Tha19], and that the random injective mapping used in Shoup's model for explicitly representing group elements may potentially be exploited towards this goal. Although, this can perhaps be viewed as somewhat abusing Shoup's model by relying on the randomness provided by the injective mapping (which does not actually exist in concrete implementation of cryptographic groups) instead of relying on the algebraic structure of the group.

The efficiency of batch verification in other settings. Our work considers the efficiency of batch verification in the specific setting of accumulators. More generally, however, the efficiency of batch verification may be interesting to study in other settings as well. One such setting is the general one of non-interactive arguments, and specifically that of succinct non-interactive arguments [Mic94] (which seem tightly related to accumulators as succinct non-interactive arguments may be used to provide, for example, short membership proofs for accumulated values).

References

[ABC$^+$12] J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. In *Proceedings of the 9th Theory of Cryptography Conference*, pages 1–20, 2012.

[AHI$^+$17] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2087–2104, 2017.

[AHK20] T. Agrikola, D. Hofheinz, and J. Kastner. On instantiating the algebraic group model from falsifiable assumptions. In *Advances in Cryptology – EUROCRYPT '20*, pages 96–126, 2020.

[AM09] D. Aggarwal and U. M. Maurer. Breaking RSA generically is equivalent to factoring. In *Advances in Cryptology – EUROCRYPT '09*, pages 36–53, 2009.

[BBF19] D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In *Advances in Cryptology – CRYPTO '19*, pages 561–586, 2019.

[BCR$^+$19] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology – EUROCRYPT '19*, pages 103–128, 2019.

[BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner. Interactive oracle proofs. In *Proceedings of the 14th Theory of Cryptography Conference*, pages 31–60, 2016.

[BdM93] J. C. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology – EUROCRYPT '93*, pages 274–285, 1993.

[BGM$^+$92] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation. In *Advances in Cryptology – EUROCRYPT '92*, pages 200–207, 1992.

[BL96] D. Boneh and R. J. Lipton. Algorithms for black-box fields and their application to cryptography. In *Advances in Cryptology – CRYPTO '96*, pages 283–297, 1996.

[BLL00] A. Buldas, P. Laud, and H. Lipmaa. Accountable certificate management using undeniable attestations. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 9–17, 2000.

[BM07] B. Barak and M. Mahmoody-Ghidary. Lower bounds on signatures from symmetric primitives. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 680–688, 2007.

[BM09] B. Barak and M. Mahmoody-Ghidary. Merkle puzzles are optimal - An $O(n^2)$-query attack on any key exchange from a random oracle. In *Advances in Cryptology – CRYPTO '09*, pages 374–390, 2009.

[BP97] N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology – EUROCRYPT '97*, pages 480–494, 1997.

[BSBH$^+$18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. `https://eprint.iacr.org/2018/046`.

[CF13] D. Catalano and D. Fiore. Vector commitments and their applications. In *Proceedings of the 16th International Conference on Practice and Theory in Public-Key Cryptography*, pages 55–72, 2013.

[CF14] S. Y. Conner Fromknecht, Dragos Velicanu. A decentralized public key infrastructure with identity retention. Cryptology ePrint Archive, Report 2014/803, 2014. `https://eprint.iacr.org/2014/803`.

[CHK$^+$08] P. Camacho, A. Hevia, M. A. Kiwi, and R. Opazo. Strong accumulators from collision-resistant hashing. In *Proceedings of the 11th International Conference on Information Security*, pages 471–486, 2008.

[CJ10] S. Canard and A. Jambert. On extended sanitizable signature schemes. In *Topics in Cryptology – CT-RSA '10*, pages 179–194, 2010.

[CKS09] J. Camenisch, M. Kohlweiss, and C. Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography*, pages 481–500, 2009.

[CL02] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology – CRYPTO '02*, pages 61–76, 2002.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DK02] I. Damgård and M. Koprowski. Generic lower bounds for root extraction and signature schemes in general groups. In *Advances in Cryptology – EUROCRYPT '02*, pages 256–271, 2002.

[DT08] I. Damgård and N. Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008. `https://eprint.iacr.org/2008/538`.

[FKL18] G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *Advances in Cryptology – CRYPTO '18*, pages 33–62, 2018.

[FPS20] G. Fuchsbauer, A. Plouviez, and Y. Seurin. Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In *Advances in Cryptology – EUROCRYPT '20*, pages 63–95, 2020.

[GGK$^+$05] R. Gennaro, Y. Gertner, J. Katz, and L. Trevisan. Bounds on the efficiency of generic cryptographic constructions. *SIAM Journal on Computing*, 35(1):217–246, 2005.

[GGM14] C. Garman, M. Green, and I. Miers. Decentralized anonymous credentials. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*, 2014.

[HHR$^+$15] I. Haitner, J. J. Hoch, O. Reingold, and G. Segev. Finding collisions in interactive protocols – Tight lower bounds on the round and communication complexities of statistically hiding commitments. *SIAM Journal on Computing*, 44(1):193–242, 2015.

[HK10] O. Horvitz and J. Katz. Bounds on the efficiency of black-box commitment schemes. *Theor. Comput. Sci.*, 411(10):1251–1260, 2010.

[IR89] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 44–61, 1989.

[JR10] T. Jager and A. Rupp. The semi-generic group model and applications to pairing-based cryptography. In *Advances in Cryptology – ASIACRYPT '10*, pages 539–556, 2010.

[JS08] T. Jager and J. Schwenk. On the equivalence of generic group models. In *Proceedings of the 2nd International Conference on Provable Security*, pages 200–209, 2008.

[JS13] T. Jager and J. Schwenk. On the analysis of cryptographic assumptions in the generic ring model. *Journal of Cryptology*, 26(2):225–245, 2013.

[KST99] J. H. Kim, D. R. Simon, and P. Tetali. Limits on the efficiency of one-way permutation-based hash functions. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 535–542, 1999.

[Lip12] H. Lipmaa. Secure accumulators from euclidean rings without trusted setup. In *Proceedings of the 10th International Conference on Applied Cryptography and Network Security*, pages 224–240, 2012.

[LLX07] J. Li, N. Li, and R. Xue. Universal accumulators with efficient nonmembership proofs. In *Proceedings of the 5th International Conference on Applied Cryptography and Network Security*, pages 253–269, 2007.

[LM19] R. W. F. Lai and G. Malavolta. Subvector commitments with application to succinct arguments. In *Advances in Cryptology – CRYPTO '19*, pages 530–560, 2019.

[Mau05] U. Maurer. Abstract models of computation in cryptography. In *Proceedings of the 10th IMA International Conference on Cryptography and Coding*, pages 1–12, 2005.

[Mer87] R. C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology – CRYPTO '87*, pages 369–378, 1987.

[MGG$^+$13] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Proceeding of the 2013 IEEE Symposium on Security and Privacy*, pages 397–411, 2013.

[Mic94] S. Micali. CS proofs. In *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, pages 436–453, 1994.

[MTT19] T. Mizuide, A. Takayasu, and T. Takagi. Tight reductions for Diffie-Hellman variants in the algebraic group model. In *Topics in Cryptology – CT-RSA '19*, pages 169–188, 2019.

[MW98] U. M. Maurer and S. Wolf. Lower bounds on generic algorithms in groups. In *Advances in Cryptology – EUROCRYPT '98*, pages 72–84, 1998.

[Nec94] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):91–101, 1994.

[Ngu05] L. Nguyen. Accumulators from bilinear pairings and applications. In *Topics in Cryptology – CT-RSA '05*, pages 275–292, 2005.

[NN98] K. Nissim and M. Naor. Certificate revocation and certificate update. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[OWW$^+$20] A. Ozdemir, R. S. Wahby, B. Whitehat, and D. Boneh. Scaling verifiable computation using efficient set accumulators. In *Proceedings of the 29th USENIX Security Symposium*, pages 2075–2092, 2020.

[PS14] H. C. Pöhls and K. Samelin. On updatable redactable signatures. In *Proceedings of the 12th International Conference on Applied Cryptography and Network Security*, pages 457–475, 2014.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology – EUROCRYPT '97*, pages 256–266, 1997.

[Sla12] D. Slamanig. Dynamic accumulator based discretionary access control for outsourced storage with unlinkable access. In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security*, pages 215–222, 2012.

[ST99] T. Sander and A. Ta-Shma. Flow control: A new approach for anonymity control in electronic cash systems. In *Proceedings of the 3rd International Conference on Financial Cryptography*, pages 46–61, 1999.

[Tha19] S. Thakur. Batching non-membership proofs with bilinear accumulators. Cryptology ePrint Archive, Report 2019/1147, 2019. `https://eprint.iacr.org/2019/1147`.

[Tod16] P. Todd. Making UTXO set growth irrelevant with low-latency delayed TXO commitments. Available at `https://petertodd.org/2016/delayed-txo-commitments`, 2016.

[Wee07] H. Wee. One-way permutations, interactive hashing and statistically hiding commitments. In *Proceedings of the 4th Theory of Cryptography Conference*, pages 419–433, 2007.

[Wes19] B. Wesolowski. Efficient verifiable delay functions. In *Advances in Cryptology – EUROCRYPT '19*, pages 379–407, 2019.